Redes Neuronales y Aprendizaje profundo

Tema 1.1 – Introducción al Aprendizaje Profundo

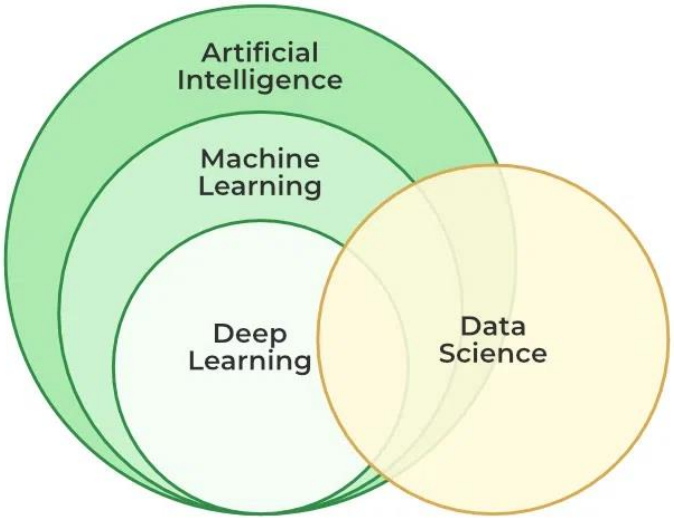
Irina Arévalo

Contenido

1. Machine Learning
2. Perceptrón
3. Redes neuronales
4. Métodos del gradiente
5. Regularización

1. Machine Learning

- Deep Learning es un subconjunto de Machine Learning



1. Machine Learning

- La principal herramienta de machine learning son los datos. En general dispondremos de **datos tabulares** en los que **cada línea representa una observación y las columna son sus características**. Depende del problema que queramos resolver, también tendremos una **columna objetivo**

Name	Pclass	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	Survived
Mr. Owen Harris Braund	3	male	22	1	0	7.25	0
Mrs. John Bradley (Florence Briggs Thayer) Cumings	1	female	38	1	0	71.2833	1
Miss. Laina Heikkinen	3	female	26	0	0	7.925	1
Mrs. Jacques Heath (Lily May Peel) Futrelle	1	female	35	1	0	53.1	1
Mr. William Henry Allen	3	male	35	0	0	8.05	0
Mr. James Moran	3	male	27	0	0	8.4583	0

- Más adelante hablaremos de otros tipos de datos, como imágenes y texto

1. Machine Learning

- Un algoritmo de machine learning es un **algoritmo que es capaz de aprender de los datos**. Para ello, buscamos una **función *f* que modelice el problema**, es decir, que pase de manera óptima por los datos que conocemos y sea capaz de generalizar (=predecir).

- Para ello, se necesitan tres componentes:

    1. Un **proceso de decisión o tarea**: el objetivo a cumplir
    2. Una **función de coste**: una manera de evaluar el algoritmo
    3. Un **algoritmo de aprendizaje**: la manera en la que tiene el modelo de aprender ajustando los parámetros del modelo. El proceso de evaluar y optimizar se itera hasta llegar a un umbral de precisión

1. Ejemplos de tareas de ML

- **Clasificación**
- **Regresión**
- Transcripción
- Traducción automática
- Detección de anomalías
- Resumen de textos
- Imputación de valores faltantes
- Generación
- Clustering
- .....

1. Clasificación

- En un problema de clasificación el objetivo es **predecir una clase binaria o categórica**.

- Ejemplos de problemas de clasificación:
  - Filtros de spam (¿es este mensaje spam? Respuestas posibles: sí o no)
  - Detección de fraude (¿es esta transacción bancaria fraudulenta? Respuestas posibles: sí o no)
  - Resultado de una quiniela (¿cuál va a ser el resultado de un partido? Respuestas posibles: 1, x, 2)
  - Análisis de sentimiento en texto (¿cuál es el ánimo del autor? Respuestas posibles: alegría, enfado, tristeza, …)

1. Regresión

- En un problema de regresión el objetivo es **predecir un output numérico.**

- Ejemplos de problemas de regresión:
  - Predecir el valor de una acción (¿cuánto valdrá ACS mañana? Respuestas posibles: cualquier valor numérico mayor que 0)
  - Predecir el valor de una casa (¿por cuánto se venderá esta casa dadas sus características? Respuestas posibles: cualquier valor numérico mayor que 0)
  - Predecir la altura de un bebé (¿cuánto medirá un bebé al nacer? Respuestas posibles: cualquier valor numérico entre 30cm y 90cm)

1. Funciones de coste

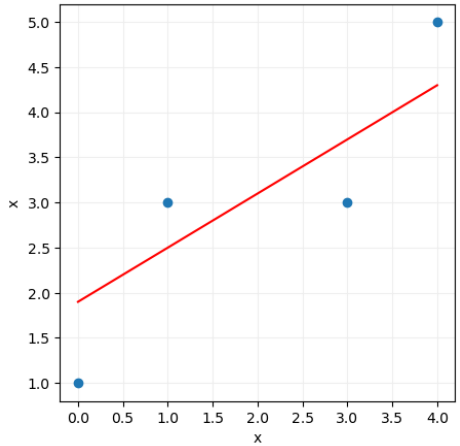
- Para problemas de clasificación y regresión, la **función de coste** (o de error o pérdida) es **una función que recibe las predicciones y el objetivo** y **devuelve** un número positivo que representa **la distancia entre ellos** (recordemos que hay muchas definiciones posibles de distancia).

- Por ejemplo, para un problema de clasificación una función de pérdida habitual es el accuracy o precisión, la proporción de observaciones para las que el modelo es acertado

- Para un problema de regresión una función de error habitual es el error cuadrático medio, el valor medio de los cuadrados de la diferencia entre los valores predichos y los valores reales

1. Funciones de coste

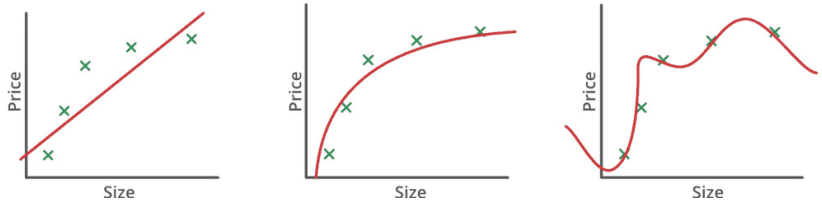
- Por ejemplo, supongamos que aproximamos nuestros datos (los puntos azules) con la función dada por la recta roja:



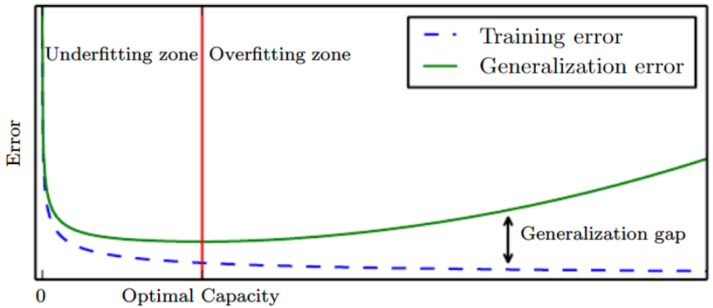
- El error cuadrático medio es 3.31, pero puede que otra recta obtenga un menor error

1. Funciones de coste

- Es importante **medir el rendimiento** del modelo en un **conjunto de datos que no haya sido usado para entrenar**, porque entonces el modelo puede no ser capaz de generalizar a datos no vistos



- Nos interesará la diferencia entre el rendimiento del modelo en los datos de entrenamiento y en los datos de test para buscar el punto óptimo



1. Algoritmo de aprendizaje

- El último requisito es una **manera de aprender**. Existen múltiples **familias de algoritmos** que **aproximan bien los datos dependiendo de sus características**.

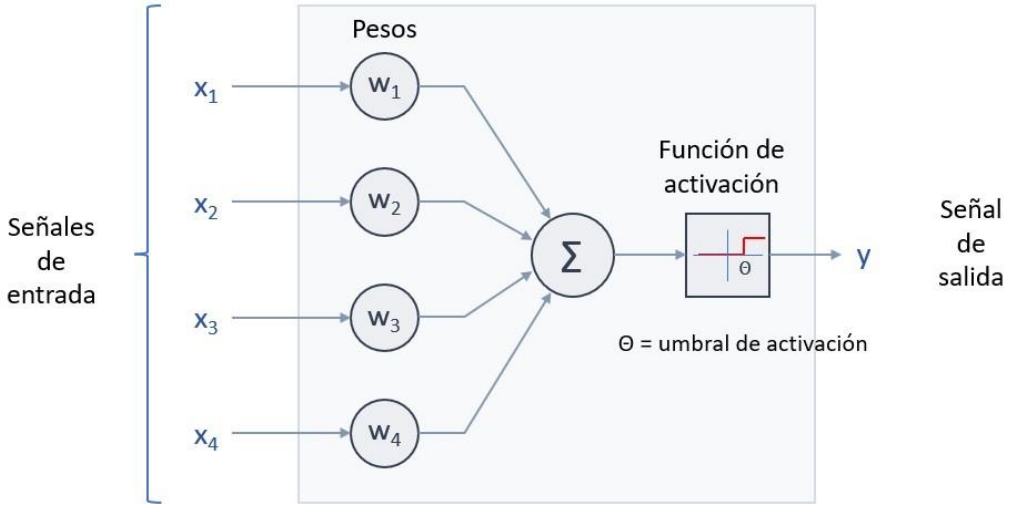
- Ejemplos de regresión:
  - Regresión lineal - aproximan bien datos en línea recta
  - Árboles de decisión - dividen el espacio de los datos de manera binaria con explicaciones simples
  - Redes neuronales - son capaces de tratar una gran cantidad de datos y características y encontrar relaciones no lineales
- Ejemplos de clasificación:
  - Regresión logística - similar a regresión lineal para problemas binarios
  - Redes neuronales

Contenido

1. Machine Learning
2. Perceptrón
3. Redes neuronales
4. Métodos del gradiente
5. Regularización

2. Perceptrón

- La primera **neurona artificial** fue desarrollada en 1943 y se conoce como Neurona de McCulloch-Pitts o **Perceptrón** (unicapa)



2. Predicción en el Perceptrón

- Al Perceptrón llega un vector $\boldsymbol{x_1, x_2, \dots, x_n}$ **de características** de una observación. **Cada $\boldsymbol{x_i}$ se multiplica por un peso $\boldsymbol{w_i}$**. En un primer paso estos pesos se inicializan de manera aleatoria, pero después el algoritmo de aprendizaje actualizará estos pesos para que aproximen mejor los datos en base a la función de coste.

- El siguiente paso al obtener los productos $x_i w_i$es sumarlos todos. Si es un problema de **regresión**, la predicción para la observación con características $x_1, x_2, \dots, x_n$ será esa suma

$$\hat{y} = \sum_{i=1}^{n} x_i w_i$$

- Si se trata de un problema de **clasificación**, la predicción será el resultado de pasar esa suma por una **función de activación** que normaliza el output entre las clases del problema de clasificación.

2. Ejemplo con perceptrón

Supongamos que tenemos los datos de Titanic:

Name	Pclass	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	Survived
Mr. Owen Harris Braund	3	0	22	1	0	7.25	0
Mrs. John Bradley (Florence Briggs Thayer) Cumings	1	1	38	1	0	71.2833	1
Miss. Laina Heikkinen	3	1	26	0	0	7.925	1
Mrs. Jacques Heath (Lily May Peel) Futrelle	1	1	35	1	0	53.1	1
Mr. William Henry Allen	3	0	35	0	0	8.05	0
Mr. James Moran	3	0	27	0	0	8.4583	0

Hemos normalizado la columna “Sex” a 0 (=male) y 1 (=female) para que sea más sencillo para el algoritmo.

Supongamos que sobre estos datos queremos responder a un **problema de clasificación**: ¿un viajero en el Titanic sobrevivió o no? Por lo tanto la **variable ”Survived” será el objetivo** (binario) de este problema.

2. Ejemplo con perceptrón

Además, la variable "Name" no da información, así que la descartamos. Así, el perceptrón tendrá **6 variables de entrada: $x_1, x_2, \dots, x_6$ y dependerá de 6 pesos $w_1, w_2, \dots, w_6$**. Por último, **la función de activación** será muy sencilla: si el valor $\sum_i x_i w_i$ es mayor que 20 (la **tolerancia** del modelo), el valor será 1 (sobrevivió) y si no, será 0 (no sobrevivió).

Supongamos que en un primer paso **inicializamos cada w_i para que valga 0.5**. Entonces la suma de la multiplicación de pesos por características de la primera observación,

Pclass	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	Survived
3	0	22	1	0	7.25	0

será $3 \cdot 0.5 + 0 \cdot 0.5 + 22 \cdot 0.5 + 1 \cdot 0.5 + 0 \cdot 0.5 + 7.25 \cdot 0.5 = 16.6$. Como es menor que 20, la predicción será 0 (no sobrevivió), que es el resultado real.

2. Ejemplo con perceptrón

Repetimos el mismo proceso con todos los datos para obtener estas predicciones:

Pclass	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	Survived	$\sum_i x_i w_i$	$\hat{y}$	Acierta?
3	0	22	1	0	7.25	0	16.625	0	Sí
1	1	38	1	0	71.2833	1	56.14165	1	Sí
3	1	26	0	0	7.925	1	18.9625	0	No
1	1	35	1	0	53.1	1	45.55	1	Sí
3	0	35	0	0	8.05	0	23.025	1	No
3	0	27	0	0	8.4583	0	19.22915	0	Sí

Con las suposiciones que hemos hecho, el **accuracy** o precisión es el porcentaje de veces que acierta (número de síes en "Acierta?" dividido por total de observaciones), que en este caso es **2/6=66.6%**. Probablemente **con otros pesos hubiésemos obtenido mejores resultados**, ya que no tiene sentido usar una inicialización aleatoria o que todas las variables tengan la misma importancia cuando tienen rangos muy dispares

2. Actualización de los pesos

- El **algoritmo de aprendizaje** habitual de un perceptrón se basa en la siguiente fórmula:

$$\Delta w_i = \eta(y - \hat{y}) \cdot x_i$$

donde y es la variable objetivo, $\hat{y}$ es la predicción y η es una constante que se llama **tasa de aprendizaje** (learning rate).

- La idea es evitar que los incrementos que se aplican a los pesos sean demasiado grandes, por lo que se suelen escoger η pequeños (por ejemplo η = 0.001), aunque también hará el aprendizaje más lento.

- Una vez encontrado cada incremento Δw_i, **actualizamos los pesos** con

$$w_i = w_i + \Delta w_i$$

2. Ejemplo con perceptrón

Recuperando el ejemplo de antes, supongamos que la tasa de aprendizaje es $\eta = 0.1$. Para la primera observación y con los pesos $w_i = 0.5$ tenemos que $y - \hat{y} = 0$, por lo que para esta observación los incrementos Δw_i son todos 0 (tiene sentido, en este caso el modelo acierta y el algoritmo de aprendizaje nos dice que no cambiemos nada).

Pclass	Sex	Age	Siblings/Spouses Aboard	Parents/Children Aboard	Fare	Survived	$\sum_i x_i w_i$	$\hat{y}$
3	0	22	1	0	7.25	0	16.625	0
1	1	38	1	0	71.2833	1	56.14165	1
3	1	26	0	0	7.925	1	18.9625	0
1	1	35	1	0	53.1	1	45.55	1
3	0	35	0	0	8.05	0	23.025	1
3	0	27	0	0	8.4583	0	19.22915	0

Para la tercera observación, para la que no acierta, los **incrementos y los nuevos pesos** son:

$$\Delta w_1 = \eta(y - \hat{y}) \cdot x_1 = 0.1 \cdot (0 - 1) \cdot 3 = -0.3 \Longrightarrow w_1 = w_1 + \Delta w_1 = 0.5 - 0.3 = 0.2$$

$$\Delta w_2 = \eta(y - \hat{y}) \cdot x_2 = 0.1 \cdot (0 - 1) \cdot 1 = -0.1 \Longrightarrow w_2 = w_2 + \Delta w_2 = 0.5 - 0.1 = 0.4$$

...

2. Actualización de los pesos

- Este proceso **continuaría con las demás muestras,** pasando una por una y **modificándose** los pesos en cada caso **si las predicciones** realizadas por la neurona **no coinciden** con la etiqueta de la muestra siendo considerada.

- El paso de todas las muestras por la neurona es lo que se conoce como un **epoch**.

- Es posible que, tras un epoch, la neurona todavía no haya convergido (es decir, es posible que los pesos no hayan tomado todavía sus valores óptimos), especialmente si la tasa de aprendizaje es pequeña, por lo que normalmente se repite el proceso durante múltiples epochs.

2. El perceptrón en Python

- La librería Scikit-learn incluye una implementación del Perceptrón para clasificación

- Vamos a entrenar un perceptrón con los datos de Titanic. Para ello, necesitamos descargar los datos e importar `Perceptron` de `sklearn`

```
import pandas as pd
import numpy as np
from sklearn.linear_model import Perceptron

df = pd.read_csv('Datos/titanic.csv’)
df['Sex'] = df['Sex'].replace({'male': 0, 'female': 1})
```

- Por costumbre llamamos X a las características de los datos e y a la variable objetivo

```
X = df.drop(["Name", "Survived"], axis = 1)
y = df["Survived"]
```

2. El perceptrón en Python

- Para entrenar un modelo de ML con sklearn primero instanciamos el modelo usando su nombre y después lo entrenamos con el comando `fit`

```
model = Perceptron()
model.fit(X, y)
```

- Por último predecimos con el método `predict` aplicado a los datos (normalizados)

```
model.predict(np.array(X.iloc[0]).reshape((1, -1)))
```

```
Output: array([0])
```

- Como es un problema de clasificación, lo evaluamos usando la función `accuracy`

```
from sklearn.metrics import accuracy_score
accuracy_score(y, model.predict(X))
```

```
Output: 0.7609921082299888
```

In [40]:

2. El perceptrón en Python

- Es importante notar que **si predecimos** en uno de los **datos con los que se ha entrenado** el modelo, los **resultados no son tan fiables** como podrían ser porque el modelo ya conocía el resultado. Lo habitual para evaluar el modelo es usar un dataset separado con datos de entrenamiento. Para ello inicialmente haríamos una **división en datos de train (normalmente un 80% de los datos) y test (el resto)**. El modelo nunca verá la variable objetivo del test.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8)
model_split = Perceptron()
model_split.fit(X_train, y_train)
accuracy_score(y_test, model_split.predict(X_test))
```

```
Output: 0.7528089887640449
```

2. El perceptrón en Python

- El método `get_params` nos devuelven los **hiperparámetros** del modelo, parámetros que podemos modificar para cambiar la configuración del modelo.

```
model.get_params()
```

```
Output: {'alpha': 0.0001, 'class_weight': None, 'early_stopping': False, 'eta0': 1.0,
'fit_intercept': True, 'tol': 0.001, …
```

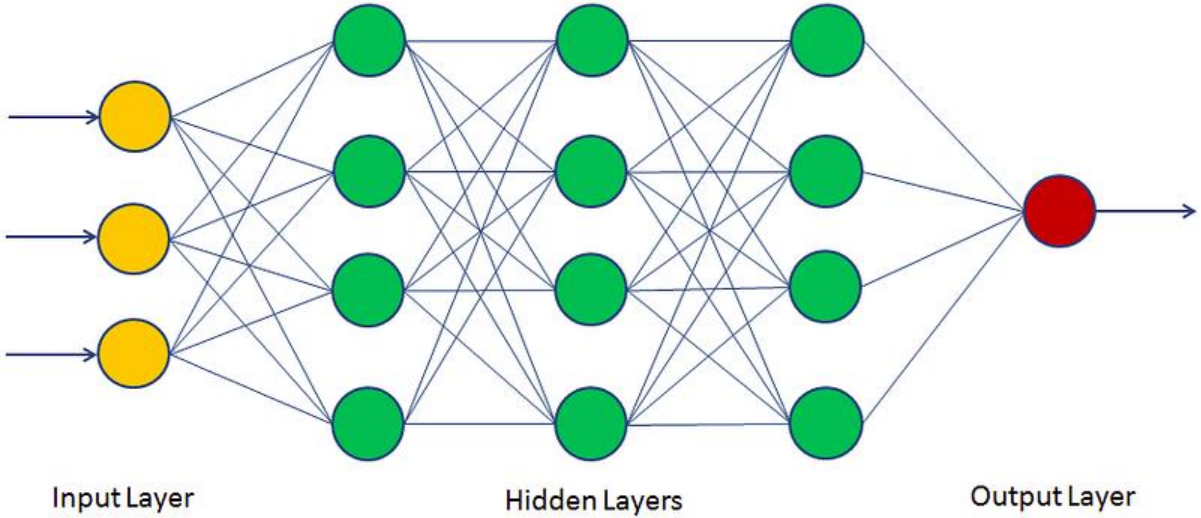
- Cambiamos cualquiera de estos parámetros en la instanciación del modelo

```
model_split = Perceptron(tol = 0.05)
model_split.fit(X_train, y_train)
accuracy_score(y_test, model_split.predict(X_test))
```

```
Output: 0.702247191011236
```

2. Perceptrón multicapa

- El **perceptrón multic**apa es una profundización del perceptrón (unicapa) en el que añadimos **una o más capas ocultas** con las mismas transformaciones lineales que para una capa. En la capa de salida se aplica una función de activación, normalmente la sigmoide para clasificación



Redes Neuronales y Aprendizaje profundo

Tema 1.1 – Introducción al Aprendizaje Profundo

Irina Arévalo